Linux RFS v1.3.0 Porting Guide

May 20-2008, Version 1.13

SAMSUNG ELECTRONICS

Copyright notice

Contact Information:

Flash Software Group

Samsung Electronics Co., Ltd
Address: San #16 BanWol- Dong, Hwasung-City,
Gyeonggi-Do, Korea, 445-701

Preface

SEC-FSG-RFS1.2-POG

This document is a porting guide of RFS developed by Flash Software Group, Memory Division, Samsung Electronics. It describes Linux RFS porting procedure to user's target platform.

Purpose

This document is RFS Porting Guide. This document explains the definition, architecture, system requirement, and porting tutorial of RFS. This document also provides the features and API of each module that a user should know well to port RFS. Combine the above two paragraphs for one into a meaningful one

Scope

This document is for Project Manager, Project Leader, Application Programmers, etc.

Definitions and Acronyms

FTL (Flash Translation Layer)	A software module which maps between logical addresses and physical addresses when accessing to flash memory
IDE	Integrated Development Environment
CRAMFS	Compressed ROM File System
RFS	Robust FAT File System
VFS	Virtual File System
XSR	eXtended Sector Remapper
MTD	Memory Technology Device
LLD	Low Level Device Driver
Sector	The file system performs read/write operations in a 512-byte unit called sector.
Page	NAND flash memory is partitioned into fixed-sized pages. A page is (512+16) bytes or (2048 + 64) bytes.
Block	NAND flash memory is partitioned into fixed-sized blocks. A block is 16K bytes or 128K bytes.
NAND flash device	NAND flash device is a device that contains NAND flash memory or NAND flash controller.
NAND flash memory	NAND-type flash memory
Deferred Check Operation	The method that can increase time and device operation performance. Every operation function of LLD defers the check routine to the next operation.
OneNAND	Samsung NAND flash device that includes NAND flash memory and NAND flash controller.

Related Documents

- RFS v1.3.0 Programmer's Guide, Samsung Electronics, Co., Ltd.
- LinuStoreII Utility Guide, Samsung Electronics Co., Ltd.
- LinuStoreII Porting Guide, Samsung Electronics Co., Ltd.

History

Version	Date	Comment	Author
0.1	2006.01.05	Initial	Flash Software Group
0.8	2006.01.13	1st draft	Flash Software Group
1.4	2006.02.13	Release	Thomas
1.5	2006.03.16	Release	Amit
1.6	2006.03.23	Release	Amit
1.7	2006.04.19	RFS1.2.1 release	Amit
1.8	2006.05.10	DPM porting added	Seung-Jin Jung
1.9	2006.09.27	NLS added	Ha-Young Kim
1.10	2006-09-29	H/W int added	Kyu-Hyung Kim
1.11	2006.12.26	Review comments	Vishu Kumar
1.12	2007.03.19	SAM factor modifications	Seung-Jin Jung
1.13	2008-05-20	Change version to 1.3.0 Delete sections about xsr	Hayoung Kim

SAMSUNG DIGITall
everyone's invited™

Table of Contents

Tables

Figures

1 Introduction

This chapter describes the overview and system architecture of RFS. It also covers the information about low-level flash memory partitions.

1.1 Overview

Linux RFS (Linux Robust FAT File System) is a FAT compatible file system to use OneNAND/NAND flash memory as storage on any consumer electronic devices. As its name implies, Linux RFS runs in the Linux kernel and is fully compatible with FAT file system standards (FAT16/32). For 'robustness', it provides a journaling based error recovery mechanism, which guarantees that the file system runs at all times even if there is a sudden power loss.

Currently, there are a few open-source projects that implement NAND based flash file systems such as JFFS2, YAFFS and YAFFS2. However, they have a limited applicability to brand-new OneNAND flash memory devices that feature advanced technologies to improve performance.

RFS supports all of the OneNAND flash memory devices in the market and is optimized for those devices. RFS can deliver much better read/write performance compared to the existing solutions.

1.2 Features

The following are RFS features.

- FAT compatible file system which supports FAT16/32
- Robust error-recovery system based on journaling
- Provide POSIX-compatible interface
- Supports Linux kernel 2.4.X and 2.6.X (tested on MontaVista Linux)
- Supports demand paging
- Packaged as a loadable kernel module

1.3 Architecture

Figure 1-1 shows typical file system architecture where RFS is used for OneNAND flash memory. Linux RFS flash file system has been divided into different layers based on their operations. It consists of following layers: FAT file system layer, flash block device driver layer, sector translation layer, block management layer, and low level device driver layer.

Low level device driver layer code for OneNAND^{TM} is provided with RFS package and is tested on OMAP2420 board. For other chipsets, you need to write your own low level driver code. A brief description about each of these components is given here.

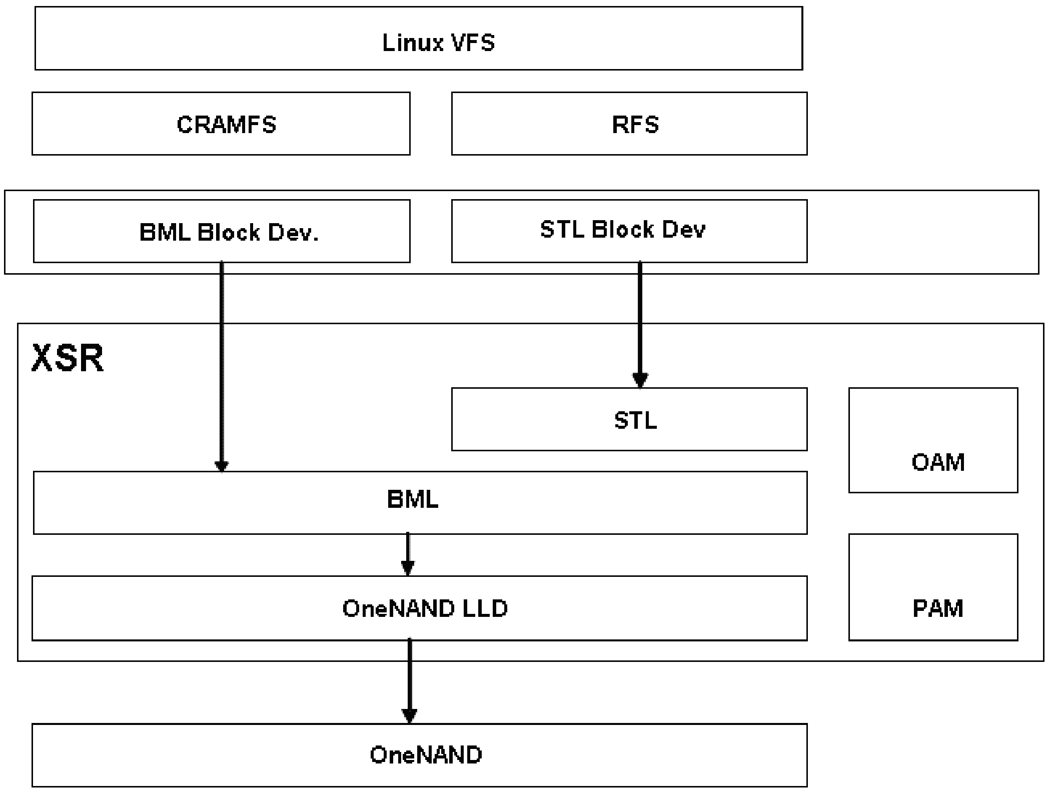


Figure 1-1 Linux RFS Architecture

☐ **File Systems**

Linux file system for flash devices is managed by two file systems: CRAMFS and RFS. Both of these file systems run under Linux VFS (Virtual File System).

- **CRAMFS:** This is a read-only file system included in a standard Linux kernel distribution. CRAMFS is used to manage the read-only part of Linux file system directories, i.e. execution code, libraries, and ROM data files. Because CRAMFS supports run-time code decompression, you can store code in a compressed form, which results in high space efficiency. CRAMFS runs on top of the BML Block Device Driver.

- **RFS:** This is a read-write Samsung's file system and is used to manage the modifiable data part of the Linux file system directories, i.e. configuration files and applications data files. It does not support compression feature. RFS runs on top of the STL Block Device Driver.

☐ **XSR Block Device Drivers**

Block device drivers are used to provide common block device APIs to file systems. For each of the block device drivers, separate block device files provide device interface methods like mount and format to applications. RFS needs two kinds of block device driver interface.

- **BML Block Device Driver:** This block device driver is used to provide driver functions for the device files /dev/bml0/*. This block device driver acts right upon the BML. There is no logical to physical address translation because there is no FTL-like software layer between BML and this driver layer. For example, logical sector number n directly maps to physical sector number n. Data write to a sector involves following sequence of low-level flash operations:

    1. Block copy for back-up
    2. Block erase
    3. Copy back for non-modified pages

4. Writing the sector data to the modified page

These sequences of operations are not atomic, so a write request to this block device driver is prone to data corruption. For this reason, read-only file systems such as CRAMFS are adequate to run on top of this block device driver.

• **STL Block Device Driver:** This block device driver is used to provide driver functions for the device files /dev/stl0/*, /dev/stl1/* and so on. Since there is FTL between this block device driver and BML, it is allowed to perform random write requests and write requests are handled atomically. Thus any read-write file system (e.g. RFS) can run on this block device driver.

☐ XSR core

XSR core is composed of two layers: STL (Sector Translation Layer) and BML (Block Management Layer). STL is a top layer of XSR. BML is below the STL. These layers have different features and jointly provide block device interface to upper layer. The main features of each layer are as follows.

• **STL** (Sector Translation Layer): translates a logical address from the file system into the virtual flash address. It internally has wear-leveling[1] during the address translation.
• **BML** (Block Management Layer): translates the virtual address from the upper layer into the physical address. At this time, BML does the address translation in consideration of bad block and the number of NAND device in use. BML accesses LLD[2], which actually performs read, write, or erase operation, with the physical address.

☞ **Note**

Each layer of XSR can be operated separately as a module. Thus, STL can be used with other layer, which has same functionalities with BML.

☐ OAM (OS Adaptation Module)

OAM is at the right of the figure. OAM connects XSR with the OS. OAM needs to be configured according to your OS environment to use NAND flash memory. OAM module is already configured with RFS for Linux.

☐ PAM (Platform Adaptation Module)

PAM is below OAM. PAM connects XSR with the platform. PAM also needs to be configured according to your platform to use NAND flash memory.

☐ LLD (Low Level Driver)

There is a low level device driver between BML and NAND flash memory. It reads, writes, or erases data on the physical sector address received from XSR and is controlled by BML.

■ [1] Wear-leveling is an internal operation to use every block of NAND flash memory evenly through the algorithm. It extends NAND flash memory life span.
■ [2] LLD is an abbreviation of Low Level Device Driver. It performs actual read/write/erase operation to NAND flash memory as a device driver.

2 Prerequisites

This chapter explains the host/target system environment for porting RFS to your target system. Host is Linux PC environment and target can be any kind of consumer device using OneNAND flash memory.

2.1 Host Environment

The following table shows the host system requirements for configuring and building RFS 1.2

Table 1 Host System Requirements

Host Machine	PC
Host OS	Linux
IDE & Compiler	Native GCC compiler & Cross-Compiler
Free Space	About 50MB

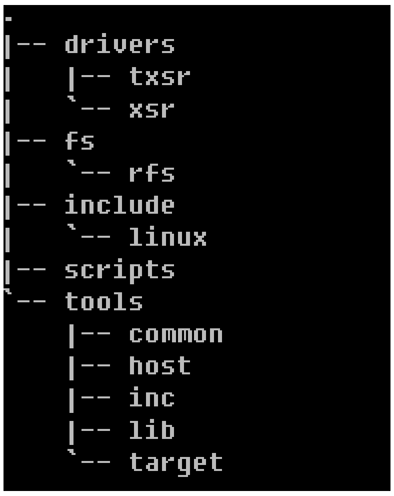
2.2 Software Environment

2.2.1 Directory Structure of Linux RFS Package

You can make a RFS directory to be the top directory of this project and extract source files from the package file using the shell command. It is also assumed that the $(TOP_DIR) also contains the Linux kernel source directory where RFS will be applied.

```
shell> cd $(TOP_DIR)
shell> tar xvjf rfs-1.3.x.tar.bz2
```

Then, some directories such as RFS, RFS-TOOLS, etc are created under $(RFS_TOP_DIR). There are RFS source, library and some tools. The RFS source package has the following directory hierarchy.



```
.
|-- drivers
|   |-- txsr
|   `-- xsr
|-- fs
|   `-- rfs
|-- include
|   `-- linux
|-- scripts
`-- tools
    |-- common
    |-- host
    |-- inc
    |-- lib
    `-- target
```

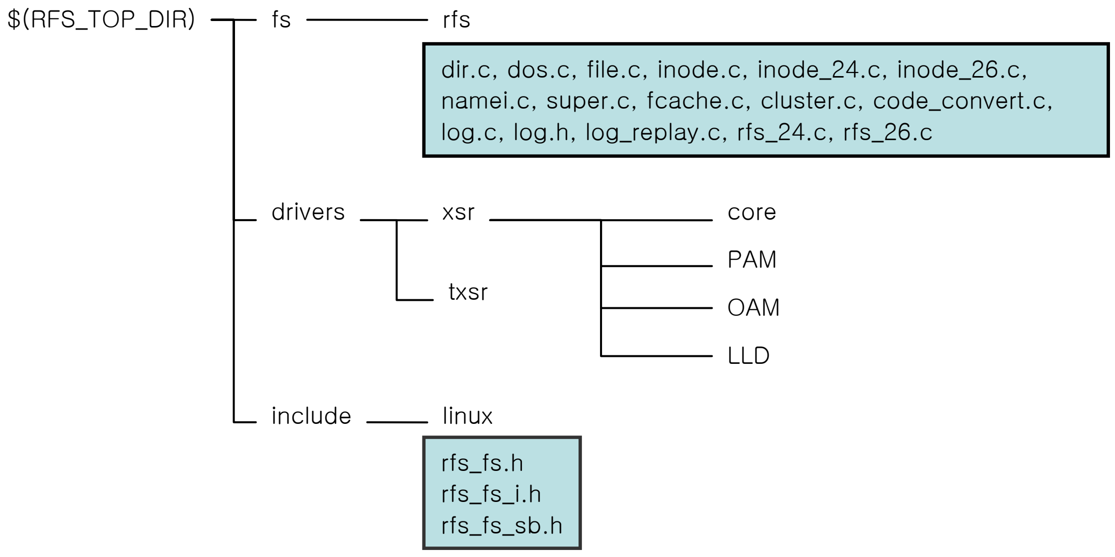
Figure 2-1 Directory Structure of Linux RFS Package

'rfs' contains source files related to Robust FAT and 'xsr' contains source files related to OneNAND block device driver. 'scripts' contains install scripts and 'util' contains several tools to maintain RFS.

- ☐ **fs** – RFS file system module
- ☐ **drivers -** XSR block device driver module
- ☐ **tools -** Utilities to manipulate RFS
- ☐ **scripts**- RFS Package Installation scripts
- ☐ **Include -** includes header files

2.2.2 Source Files List

This section gives short description of source files listed in the 'rfs' and 'xsr' directory.
The source files in this package are listed in Figure 2-2, where *.c files and *.h files are annotated on the 'RFS' source tree. Sample LLD code of OneNAND is tested with Apollon customized board based on OMAP2420 core. You need to write your own code for other chipset.



$(RFS_TOP_DIR)
fs
rfs
dir.c, dos.c, file.c, inode.c, inode_24.c, inode_26.c, namei.c, super.c, fcache.c, cluster.c, code_convert.c, log.c, log.h, log_replay.c, rfs_24.c, rfs_26.c
drivers
xsr
core
PAM
txsr
OAM
LLD
include
linux
rfs_fs.h
rfs_fs_i.h
rfs_fs_sb.h

Figure 2-2 Linux RFS Source Files (Annotated on the Source Tree)

Here are brief descriptions about *.c files.

☐ $(RFS_TOP_DIR)/fs/rfs: FAT and logging for reliability

- fcache.c: FAT cache & FAT entry handling functions
- cluster.c: FAT cluster & FAT table handling functions
- code_convert.c: Dos name and Unicode name handling operations
- dir.c: Directory handling functions
- dos.c: FAT directory entry Manipulation and management operations
- file.c: File and file inode functions
- inode.c: Common inode operations
- inode_24.c: Kernel version 2.4 specific inode functions
- inode_26.c: Kernel version 2.6 specific inode functions
- log.c: Functions for logging
- log_replay.c: Functions for replaying log
- namei.c: Adaptation layer between the VFS and RFS file system for inode operations
- rfs_24.c: Kernel version 2.4 specific functions
- rfs_26.c: Kernel version 2.6 specific functions
- super.c: Super block and init functions

☐ $(RFS_TOP_DIR)/drives/xsr: NAND block device driver for RFS

☐ $(RFS_TOP_DIR)/tools/: Tools to manipulate RFS/XSR

Most of the sources are platform-independent codes except PAM. Please refer the "XSR Porting Guide" for more detailed information. Before going into detail about RFS customization according to target requirement, next section will explain about objects that Linux RFS will create during make (or build) steps.

2.2.3 RFS Memory Usage

Table 2 lists the RFS static memory usage.

Table 2 RFS Static Memory Usage (in bytes)

Module	TEXT	DATA	BSS	Total
RFS	45,112	680	4	45,796
XSR	80,808	2,692	4,288	87,788
Sum	125,920	3,372	4,292	133,584

2.3 Hardware Environment

In this porting guide, OMAP2420 is used as target board to give porting example of RFS. 3 shows hardware information about OMAP2420.

Table 3 shows hardware information about OMA2420.

Table 3 Hardware information of OMAP2420

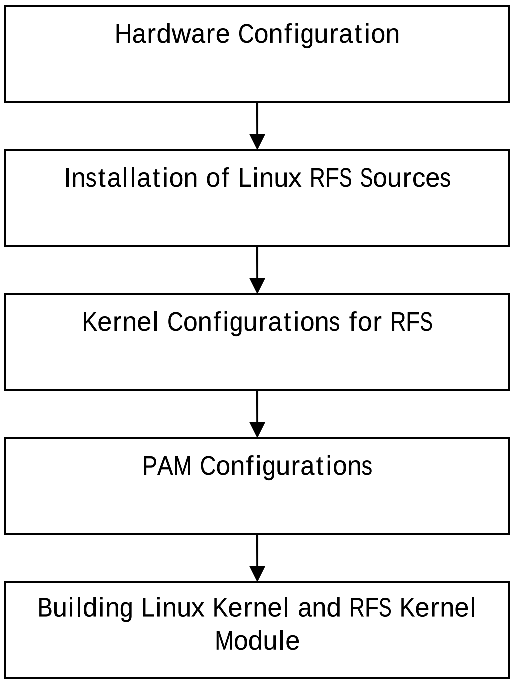
CPU	ARM1136JF-S core and TMS320C55x DSP core
Memory	SRAM: 640K bytes of shared inetrnal RAM
External Memory Interface	1. General Purpose Memory Controller (GPMC) Up to 100MHz, NOR flash, NAND flash, SRAM, and PSRAM asynchronous and synchronous protocols 16-bit data, up to eight chip-selects, 128M-byte address bus, 1G-byte total address space 2. SDRAM controller (SDRC) SDRAM, DDR mobile SDRAM, mobile DDR 16- or 32-bit data, two chip-selects, configurations up to 2G bits on each chip-select 16bit
MPU Peripheral	v6 instruction set architecture (ISA), 32K-byte instruction and 3K-byte data 64-entry instruction and 64-entry data write buffer Vector floating-point processor Jazelle Java accelerator
OneNAND device	Select OneNAND device according to your requirement (1.8v/3.3v)

3 Porting Linux RFS

This chapter describes porting overview, hardware configuration for OneNAND, Linux RFS initialization and porting procedure with OMAP2420 target board.

3.1 Porting Overview

This section describes Linux RFS porting procedure briefly. The procedure is divided into 5 steps as shown in the following figure.



Hardware Configuration
Installation of Linux RFS Sources
Kernel Configurations for RFS
PAM Configurations
Building Linux Kernel and RFS Kernel Module

Figure 3-1 Linux RFS Porting Procedure

These steps will guide you in porting RFS on your target Platform.

3.2 Porting Procedure

This section will explain you in detail about porting procedure.

3.2.1 Installation of Linux RFS Sources

The first thing you should do is to install the Linux RFS sources into the target system's kernel source tree. This step is very easy to fulfill. Extract Linux RFS source files from the package file on your Linux host PC as explained in Section 2.1. Now go inside the following folder $(TOP_DIR)/$(RFS_TOP_DIR)/scripts/. In this folder you will find file rfs_install.sh, open this file and edit following variable in this file.

KERNEL	= Set kernel path
RFS	= Set RFS source top directory path
ARCH	= Set the architecture type

Now run "scripts/rfs_install.sh [kernel_type]" at $(TOP_DIR)/

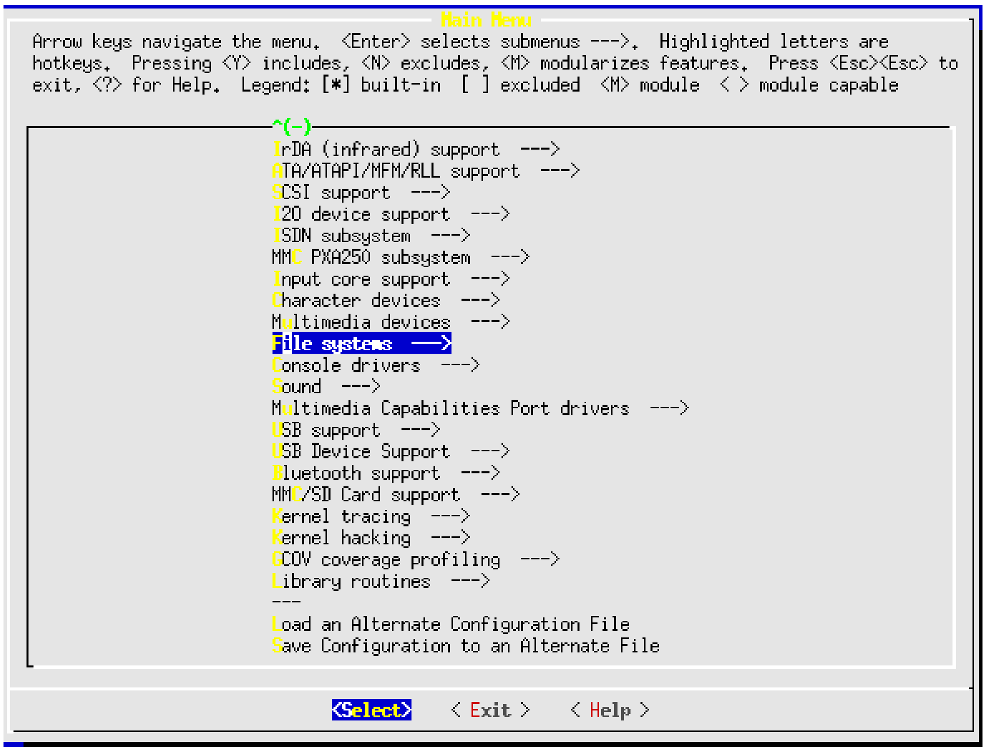
If you are using Linux kernel 2.4.xx specify ‘kernel type’ as 24. If you are using Linux kernel 2.6.xx specify ‘kernel type’ as 26.

```
Shell> cd $(TOP_DIR)
Shell> $(RFS_TOP_DIR)/scripts/rfs_install.sh 24
```

3.2.2 Kernel Configuration for RFS

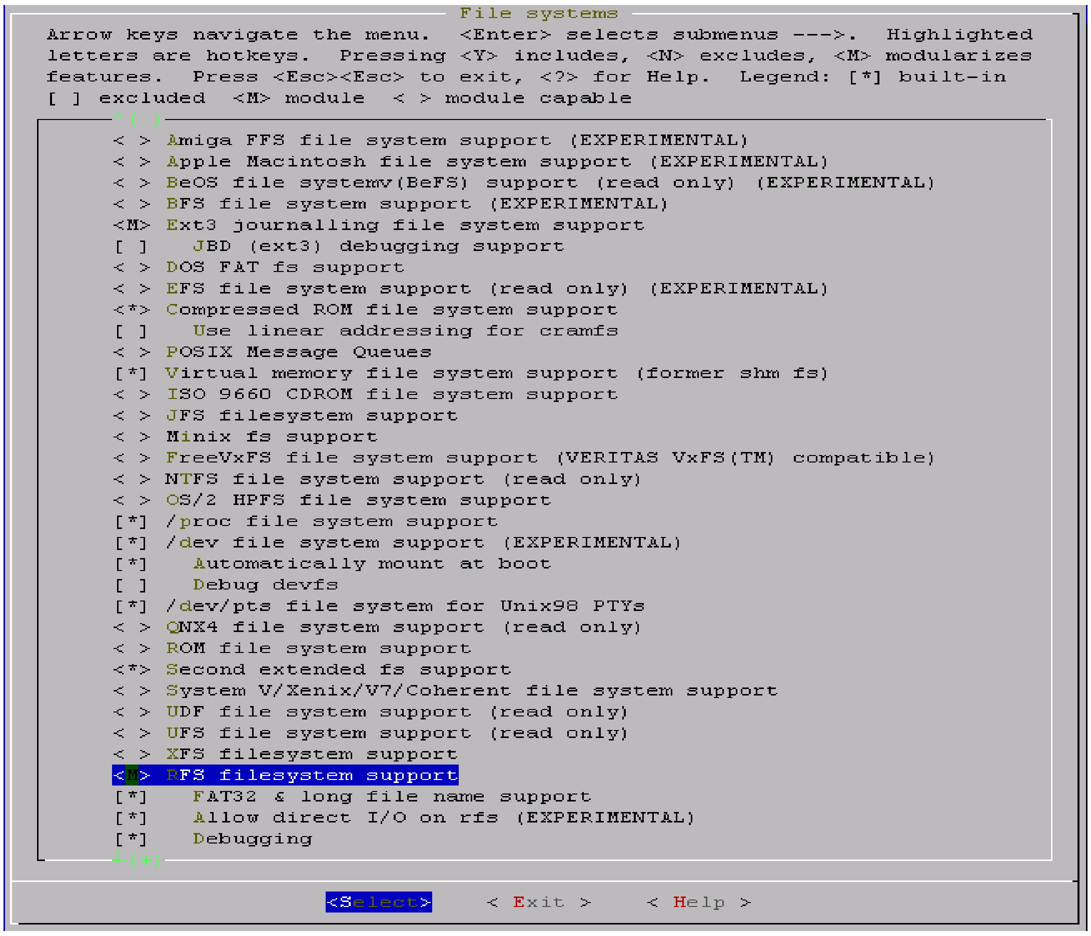
3.2.2.1 Menu Configuration of Kernel for RFS

As shown below, you should get sub-menu in $(KERNEL_TOP_DIR) as the result of ‘make menuconfig’. Figure 3-2 shows the general main menu of kernel configuration. Select “File systems” menu to go on to the next step.



```
                              Main Menu
  Arrow keys navigate the menu.  <Enter> selects submenus --->.  Highlighted letters are
  hotkeys.  Pressing <Y> includes, <N> excludes, <M> modularizes features.  Press <Esc><Esc> to
  exit, <?> for Help.  Legend: [*] built-in  [ ] excluded  <M> module  < > module capable

                          ^(-)
                          IrDA (infrared) support  --->
                          ATA/ATAPI/MFM/RLL support  --->
                          SCSI support  --->
                          I2O device support  --->
                          ISDN subsystem  --->
                          MMC PXA250 subsystem  --->
                          Input core support  --->
                          Character devices  --->
                          Multimedia devices  --->
                          File systems  --->
                          Console drivers  --->
                          Sound  --->
                          Multimedia Capabilities Port drivers  --->
                          USB support  --->
                          USB Device Support  --->
                          Bluetooth support  --->
                          MMC/SD Card support  --->
                          Kernel tracing  --->
                          Kernel hacking  --->
                          GCOV coverage profiling  --->
                          Library routines  --->
                          ---
                          Load an Alternate Configuration File
                          Save Configuration to an Alternate File

                            <Select>    < Exit >    < Help >
```

Figure 3-2 Main screen of Kernel menu



```
                                 File systems
   Arrow keys navigate the menu.  <Enter> selects submenus --->.  Highlighted
   letters are hotkeys.  Pressing <Y> includes, <N> excludes, <M> modularizes
   features.  Press <Esc><Esc> to exit, <?> for Help.  Legend: [*] built-in
   [ ] excluded  <M> module  < > module capable

        ^(-)
        < > Amiga FFS file system support (EXPERIMENTAL)
        < > Apple Macintosh file system support (EXPERIMENTAL)
        < > BeOS file systemv(BeFS) support (read only) (EXPERIMENTAL)
        < > BFS file system support (EXPERIMENTAL)
        <M> Ext3 journalling file system support
        [ ]   JBD (ext3) debugging support
        < > DOS FAT fs support
        < > EFS file system support (read only) (EXPERIMENTAL)
        <*> Compressed ROM file system support
        [ ]   Use linear addressing for cramfs
        < > POSIX Message Queues
        [*] Virtual memory file system support (former shm fs)
        < > ISO 9660 CDROM file system support
        < > JFS filesystem support
        < > Minix fs support
        < > FreeVxFS file system support (VERITAS VxFS(TM) compatible)
        < > NTFS file system support (read only)
        < > OS/2 HPFS file system support
        [*] /proc file system support
        [*] /dev file system support (EXPERIMENTAL)
        [*]   Automatically mount at boot
        [ ]   Debug devfs
        [*] /dev/pts file system for Unix98 PTYs
        < > QNX4 file system support (read only)
        < > ROM file system support
        <*> Second extended fs support
        < > System V/Xenix/V7/Coherent file system support
        < > UDF file system support (read only)
        < > UFS file system support (read only)
        < > XFS filesystem support
        <M> RFS filesystem support
        [*]   FAT32 & long file name support
        [*]   Allow direct I/O on rfs (EXPERIMENTAL)
        [*]   Debugging
        v(+)

                        <Select>    < Exit >    < Help >
```

Figure 3-3 File system screen of Kernel menu

Figure 3-3 shows the detailed RFS configuration.

1. The first is FAT32 and long file name support
2. The second is direct I/O support. But, this feature is experimental and should not be used in production environment.
3. The third is debugging message option

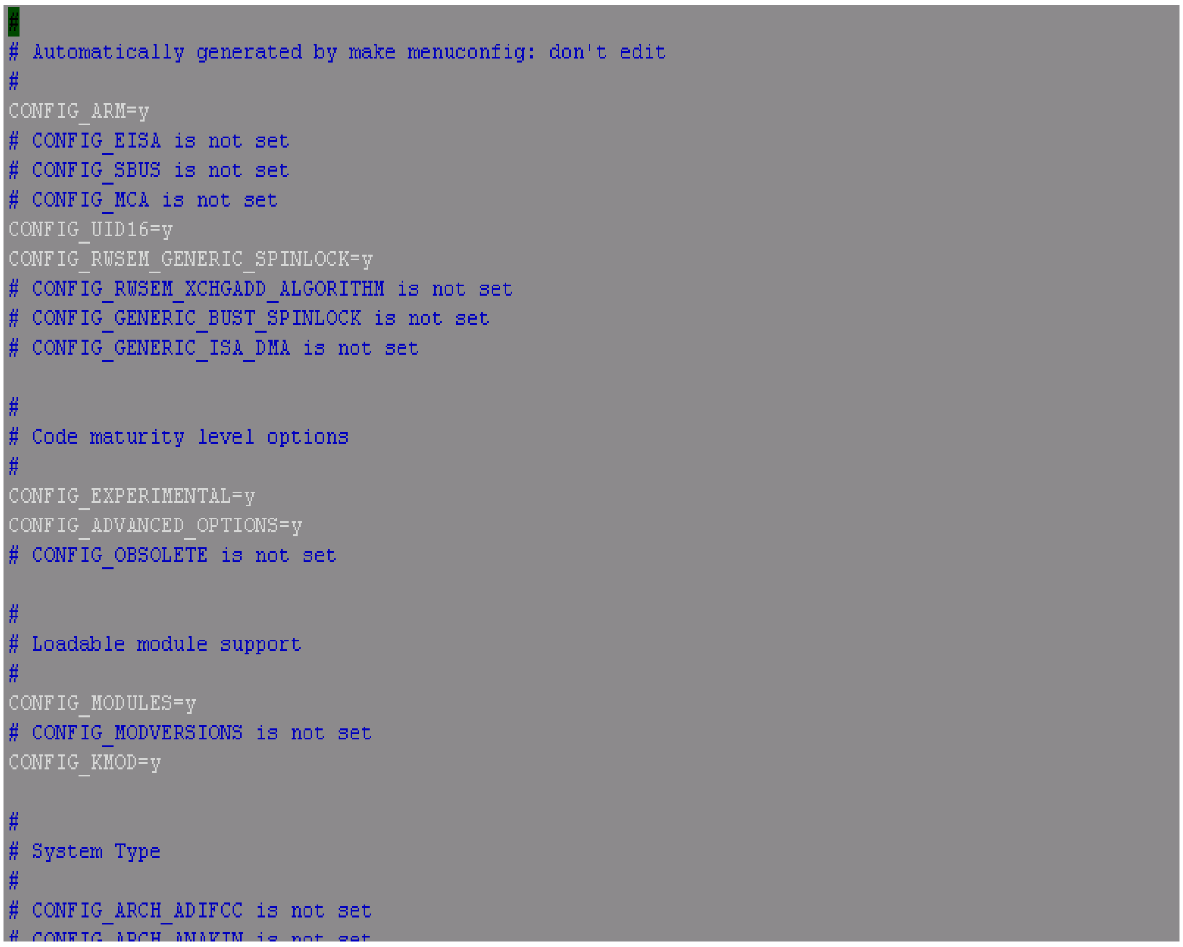
☞ **NOTE**

You have to execute the 'mkrfsnod' utility on the target for making device nodes used by XSR block device driver. The detailed usage of this program is described in Ftools Utility Guide document.

3.2.3 Building Linux Kernel and RFS Kernel Module

In this build step, the first thing you should do is to check the kernel make options. Following are the make options you should check. You can confirm the following make options typing "make menuconfig" on the shell command line.

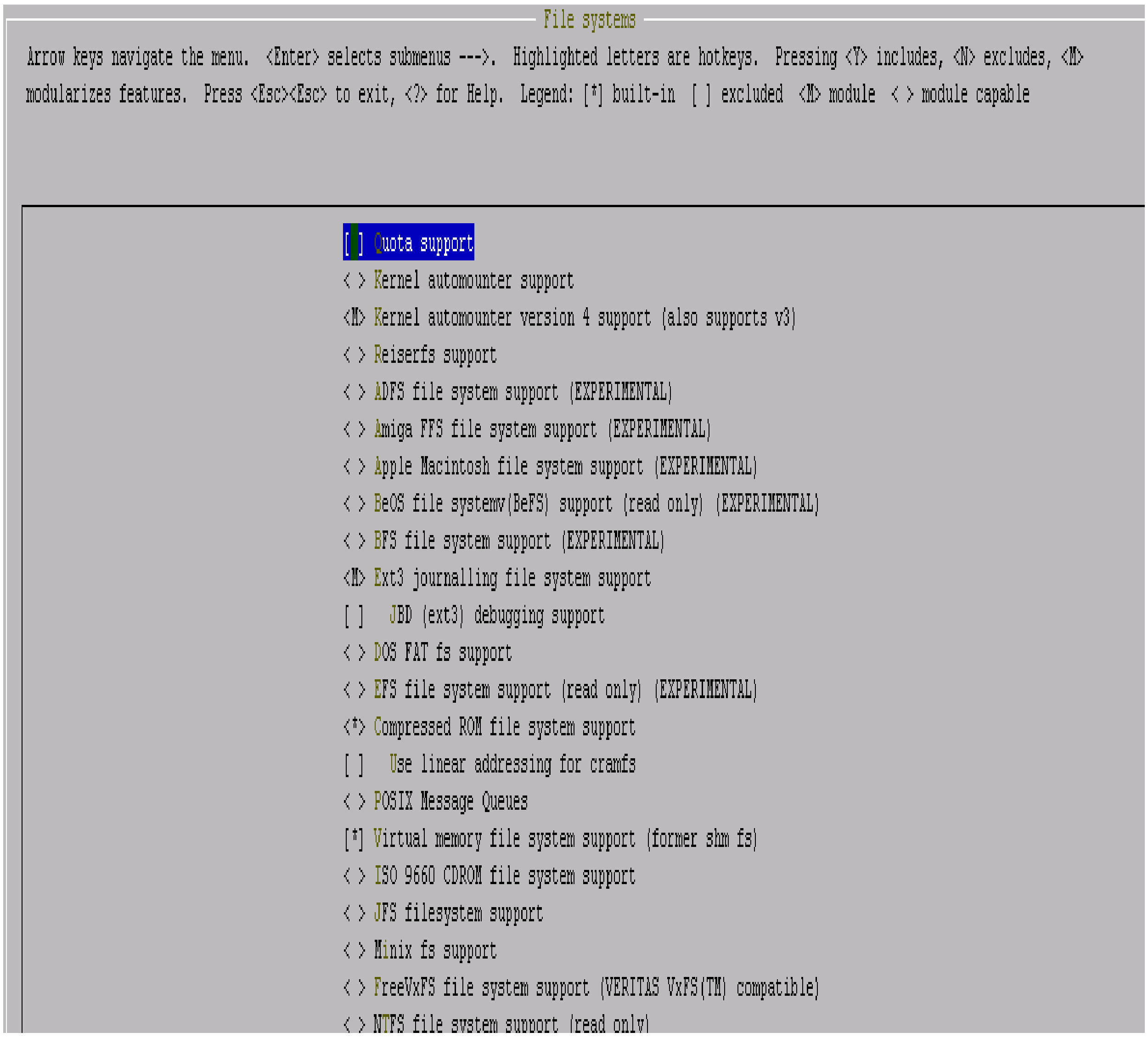
☐ You should turn on the make option for "EXPERIMENTAL" kernel features in .config file.



```
#
# Automatically generated by make menuconfig: don't edit
#
CONFIG_ARM=y
# CONFIG_EISA is not set
# CONFIG_SBUS is not set
# CONFIG_MCA is not set
CONFIG_UID16=y
CONFIG_RWSEM_GENERIC_SPINLOCK=y
# CONFIG_RWSEM_XCHGADD_ALGORITHM is not set
# CONFIG_GENERIC_BUST_SPINLOCK is not set
# CONFIG_GENERIC_ISA_DMA is not set

#
# Code maturity level options
#
CONFIG_EXPERIMENTAL=y
CONFIG_ADVANCED_OPTIONS=y
# CONFIG_OBSOLETE is not set

#
# Loadable module support
#
CONFIG_MODULES=y
# CONFIG_MODVERSIONS is not set
CONFIG_KMOD=y

#
# System Type
#
# CONFIG_ARCH_ADIFCC is not set
# CONFIG_ARCH_ANAKIN is not set
```

Figure 3-4 Code maturity level

☐ You should set the make option for “COMPRESSED ROM FILE SYSTEM (CRAMFS)” in file system option during make menuconfig because the root file system is managed by CRAMFS.



```
File systems
Arrow keys navigate the menu.  <Enter> selects submenus --->.  Highlighted letters are hotkeys.  Pressing <Y> includes, <N> excludes, <M>
modularizes features.  Press <Esc><Esc> to exit, <?> for Help.  Legend: [*] built-in  [ ] excluded  <M> module  < > module capable

[ ] Quota support
< > Kernel automounter support
<M> Kernel automounter version 4 support (also supports v3)
< > Reiserfs support
< > ADFS file system support (EXPERIMENTAL)
< > Amiga FFS file system support (EXPERIMENTAL)
< > Apple Macintosh file system support (EXPERIMENTAL)
< > BeOS file systemv(BeFS) support (read only) (EXPERIMENTAL)
< > BFS file system support (EXPERIMENTAL)
<M> Ext3 journalling file system support
[ ]   JBD (ext3) debugging support
< > DOS FAT fs support
< > EFS file system support (read only) (EXPERIMENTAL)
<*> Compressed ROM file system support
[ ]   Use linear addressing for cramfs
< > POSIX Message Queues
[*] Virtual memory file system support (former shm fs)
< > ISO 9660 CDROM file system support
< > JFS filesystem support
< > Minix fs support
< > FreeVxFS file system support (VERITAS VxFS(TM) compatible)
< > NTFS file system support (read only)
```

Figure 3-5 CRMAFS OPTION Settings

Now, you can proceed to build the kernel and the kernel modules. Before starting build process your kernel cross compile path 'CROSS_COMPILE = ' must be set in $(KERNEL_TOP_DIR)/Makefile. To build the kernel, type the following commands in sequence.

```
shell> cd $(KERNEL_TOP_DIR)
shell> make clean
shell> make dep
shell> make uImage
```

Then, the kernel image file named 'uImage' will be created if no error occurs. As mentioned earlier, the RFS sources are compiled and linked as a loadable kernel module. To make the kernel modules, do the following.

```
shell> make modules
shell> make modules_install
```

After all of these steps, you will have the kernel image 'uImage'. For usage of OneNAND device on your target, please refer to ftools utility guide.

3.3 Using the NLS (Native Language Support)

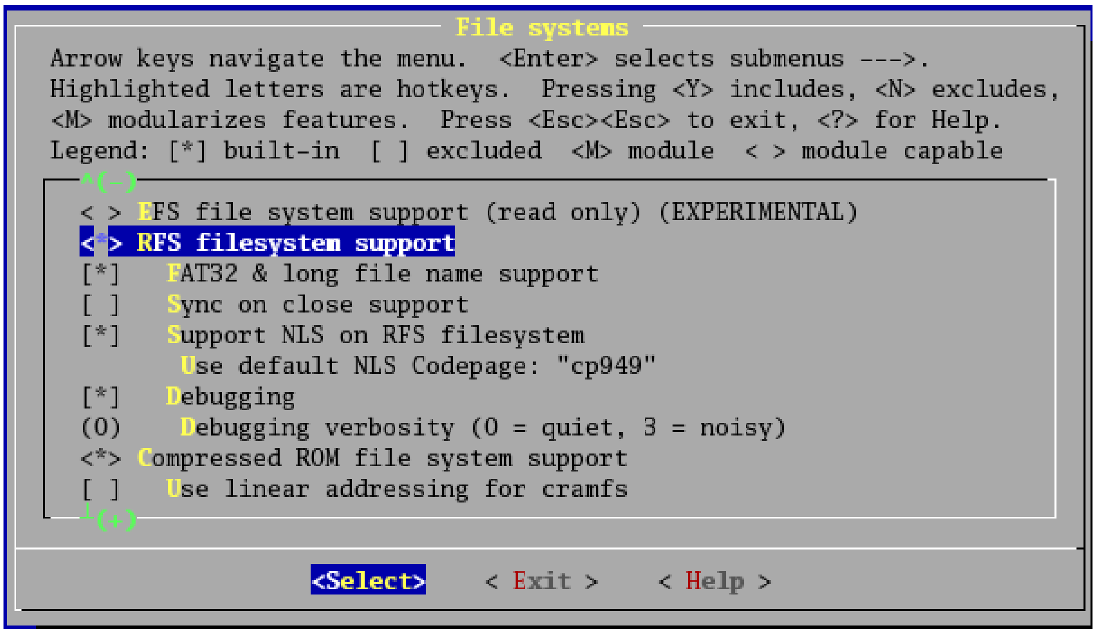
The FAT Filesystem can deal with filenames in native language character sets. These character sets are stored in so-called DOS codepages. You need to include the appropriate codepage if you want to be able to read/write these file names on DOS/Windows or other FAT partitions correctly. It applies to the filenames only, not to the file contents.

In RFS Filesystem, you can decide to use the native language for the name by the kernel configuration. So, if you don't configure the NLS option, you can make only the name with 7-bit ASCII characters.

3.3.1 Kernel Configuration for NLS

To support filenames with the native language characters, you have to set some kernel configurations like the following:

As shown below, you should select "File systems" menu at the Main menu of 'make menuconfig'.

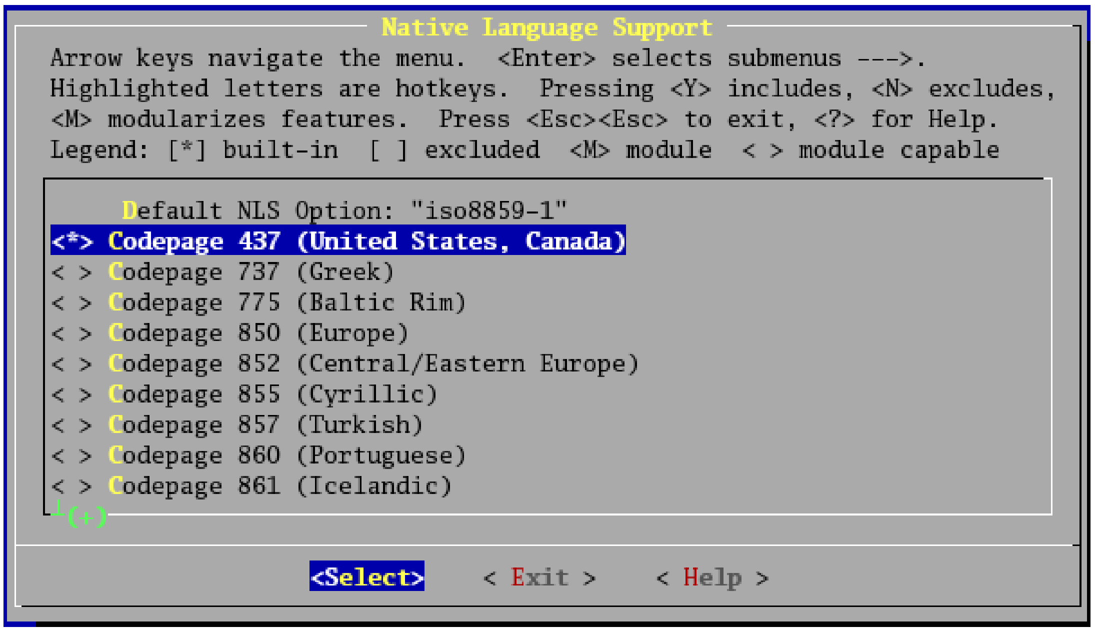


```
                          File systems
 Arrow keys navigate the menu.  <Enter> selects submenus --->.
 Highlighted letters are hotkeys.  Pressing <Y> includes, <N> excludes,
 <M> modularizes features.  Press <Esc><Esc> to exit, <?> for Help.
 Legend: [*] built-in  [ ] excluded  <M> module  < > module capable
   ^(-)
   < > EFS file system support (read only) (EXPERIMENTAL)
   <*> RFS filesystem support
   [*]   FAT32 & long file name support
   [ ]   Sync on close support
   [*]   Support NLS on RFS filesystem
          Use default NLS Codepage: "cp949"
   [*]   Debugging
   (0)    Debugging verbosity (0 = quiet, 3 = noisy)
   <*> Compressed ROM file system support
   [ ]   Use linear addressing for cramfs
   v(+)

              <Select>    < Exit >    < Help >
```

Figure 3-6 RFS Filesystem configuration for VFAT

The menu "Support NLS on RFS Filesystem" is the native language support. And if you want to support filenames with the native language, you should select this menu as <Y>.
If you select <Y>, you can set up the default codepage at the sub-menu "Use default NLS Codepage".
This default codepage is used to mount the RFS Filesystem if the "codepage" mount option is not set.

If you select <Y> for the "Support NLS on RFS Filesystem", you should select the "Native Language Support" menu at "File system" to open the NLS configuration like the following.



```
                     Native Language Support
  Arrow keys navigate the menu.  <Enter> selects submenus --->.
  Highlighted letters are hotkeys.  Pressing <Y> includes, <N> excludes,
  <M> modularizes features.  Press <Esc><Esc> to exit, <?> for Help.
  Legend: [*] built-in  [ ] excluded  <M> module  < > module capable

       Default NLS Option: "iso8859-1"
   <*> Codepage 437 (United States, Canada)
   < > Codepage 737 (Greek)
   < > Codepage 775 (Baltic Rim)
   < > Codepage 850 (Europe)
   < > Codepage 852 (Central/Eastern Europe)
   < > Codepage 855 (Cyrillic)
   < > Codepage 857 (Turkish)
   < > Codepage 860 (Portuguese)
   < > Codepage 861 (Icelandic)
   (+)

              <Select>    < Exit >    < Help >
```

Figure 3-7 NLS(Native Language Support) configuration

And you should select the codepages like the default codepage and other codepages to be used at the target. Then, the codepages will be compiled as built-in or module.
For example, if you set the default codepage to "cp949" for Korean, you have to select "Codepage 949" at this menu.

☞ **NOTE**

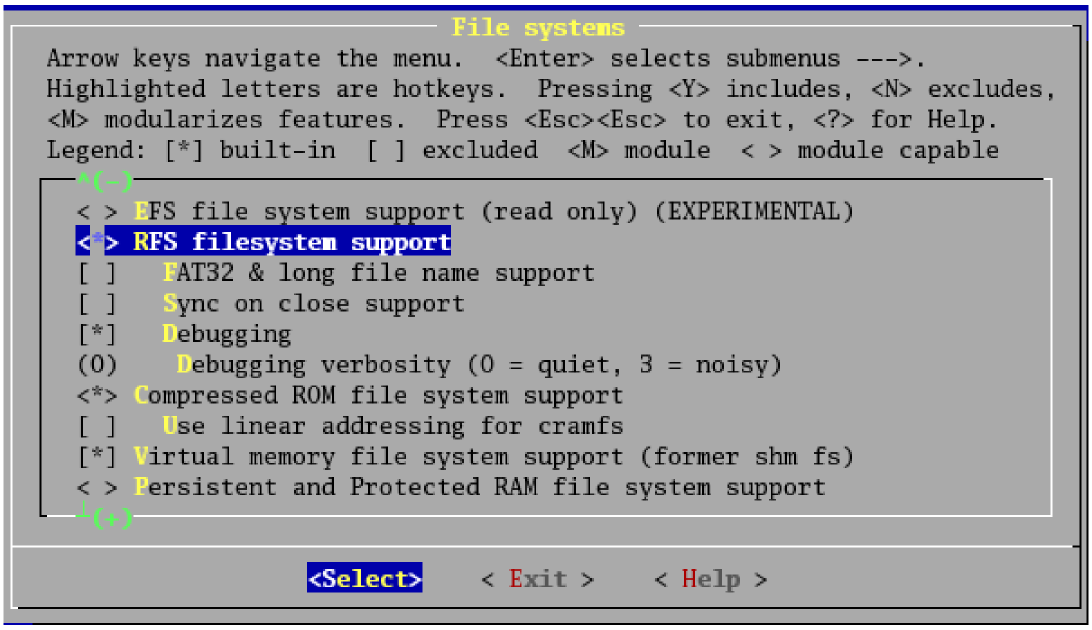
If you installed the RFS code by using 'rfs_install.sh' script, the "Native Language Support" option in the "File System" menu is automatically turned on when you turned on the option, "Support NLS on RFS Filesystem". Because that script makes some changes at the configuration file of the Filesystem.

But if you don't use 'rfs_install.sh' or "Native Language Support" option isn't turned on, you need to do the following:

► At the 2.6 kernels, you can explicitly select "Native Language Support" at the menu of "File system" and be enable to select the proper codepage.

► At the 2.4 kernels, the option "Native Language Support" is automatically selected by the Filesystem using the native language like MS-DOS, VFAT, SMB and so on. So, you have to turn on other Filesystem like MS-DOS to use the "Native Language Support".

When you didn't select "FAT32 & long file name support", you can't find the menu about "Support NLS on RFS Filesystem" like the following.



```
                         File systems
 Arrow keys navigate the menu.  <Enter> selects submenus --->.
 Highlighted letters are hotkeys.  Pressing <Y> includes, <N> excludes,
 <M> modularizes features.  Press <Esc><Esc> to exit, <?> for Help.
 Legend: [*] built-in  [ ] excluded  <M> module  < > module capable

   ^(-)
   < > EFS file system support (read only) (EXPERIMENTAL)
   <*> RFS filesystem support
   [ ]   FAT32 & long file name support
   [ ]   Sync on close support
   [*]   Debugging
   (0)    Debugging verbosity (0 = quiet, 3 = noisy)
   <*> Compressed ROM file system support
   [ ]   Use linear addressing for cramfs
   [*] Virtual memory file system support (former shm fs)
   < > Persistent and Protected RAM file system support
   v(+)

              <Select>    < Exit >    < Help >
```

Figure 3-8 RFS Filesystem configuration for FAT16

If you decide to build RFS Filesystem as FAT16 type, you always make the filenames with the native language in the 8.3 format regardless of NLS support. Why is this possible?

The NLS is used for the conversion the filenames with the native language to/from Unicode, but the conversion doesn't happen at the FAT16. So, FAT16 doesn't need any codepage.

☞ **NOTE**

If you make filenames with the native language at FAT16 and the length of the name is longer than the 8.3 format, the filename could be broken at the last character.

3.3.2 Mounting RFS with codepage

If you want to use RFS Filesystem supporting NLS, you should use "codepage" option to mount RFS Filesystem like the following.

```
Shell> mount –t rfs /dev/stl0/3 /tmp –o codepage=cp949
```

This command mounts the RFS Filesystem found on '/dev/stl0/3' at the directory '/tmp'.
If you selected the NLS support when building the kernel image for the target, the mount option "codepage=cp949" sets the default codepage as 'cp949'. And then RFS supports filenames with the native language of 'cp949' character set. If the codepage 949 isn't build at the target, this command will fail.

```
Shell> mount –t rfs /dev/stl0/3 /tmp
```

This command mounts the RFS Filesystem found on '/dev/stl0/3' at the directory '/tmp'.
If you didn't select the NLS support when you built the kernel image, this command will success and RFS is able to support filenames with the 7-bit ASCII characters only.

If you didn't select the 'FAT32 & long file name' when you built the kernel image, this command will success and RFS is able to support filenames with the native language in the 8.3 format.

If you selected the NLS support and the default codepage, this command will success if only the default codepage is configured and built. For example, if the default codepage is "codepage 437", then RFS supports filenames with the native language of 'cp437' character set.

If you selected the NLS support and didn't set the default codepage, this command will fail.

Appendix

I. Description of FAT Configuration Option

- **CONFIG_RFS_FS**

Description

Configuration option for RFS

Behavioral description

This option should be set for RFS support.

Additional notice

None

- **CONFIG_RFS_VFAT**

Description

To avoid the patent problems related to Window FAT file system, RFS can be built without codes related to Windows FAT file system.

Behavioral description

This option can be turned off by turning on the option, “FAT32 & long file name support” in the configuration menu. If you set it, you can use FAT16 which supports long file name and FAT32.

Additional notice

Once RFS is compiled with this option disabled, RFS can not be mounted as FAT32. It can only be mounted as FAT16. In addition, the features related to this option, such as a long file name and a Unicode conversion will not be available any more.

- **CONFIG_RFS_SYNC_ON_CLOSE**

Description

To support the file sync operation at file close time.

Behavioral description

The file is synchronized at close () call. When a file is opened by multiple processes, the file is synchronized at the last close () call.

Additional notice

None

- **CONFIG_RFS_NLS**

Description

To support filenames with the native language character set.

Behavioral description

This option can be selected by turning on the option, “Support NLS on RFS Filesystem” in the configuration menu, if only the option of CONFIG_RFS_VFAT is turned on. If you set this, you can use the filenames with the native language character set.

Additional notice

Once RFS is compiled with this option, RFS must be mounted with ‘codepage’ option or compiled with the default codepage.

- **CONFIG_RFS_DEFAULT_CODEPAGE**

Description

This option has the name of the default codepage.

Behavioral description

This option is valid if only the CONFIG_RFS_NLS is turned on. When the mount option 'codepage' of the RFS Filesystem is not set, this value can be used for mounting and for the conversion of the filename with this character set.

Additional notice

None

- **CONFIG_RFS_FAT_DEBUG**

Description

This option prints low-level debugging message for the RFS file system.

Behavioral description

If you set it, you can choose verbose level for debugging.

Additional notice

None

- **CONFIG_RFS_FAT_DEBUG_VERBOSE**

Description

This option determines the verbose level of the debugging.

Behavioral description

You can choose the following level.

Level	verbosity
0	Minimal
1	Audible
2	Loud
3	Noisy

Additional notice

Default debugging level is 0 that won't print anything.

- **CONFIG_RFS_MAPDESTROY**

Description

This option is to improve the performance by means of the stl map deletion.

Behavioral description

This option enables RFS to use the stl map deletion. It has dependency on the STL block device. If the STL block device is included as a static or module in kernel, it will be available. This option enables RFS to delete mappings corresponding with clusters. It deals with commands such as unlink or truncate.

Additional notice

This option is selected by default and it is not configurable.

- **CONFIG_RFS_IGET4**

Description

The iget4() kernel interface is used under the Linux kernel version 2.4.25.

Behavioral description

To get an inode, RFS uses a iget_locked() which is general kernel interface. But some kernel version don't provide a iget_locked() interface especially below the Linux kernel version 2.4.25.

This option enables RFS to use a iget4() interface instead of a iget_locked() interface. If your kernel supports a iget_locked() interface, you can disable it.

<u>Additional notice</u>

For MontaVista Linux Pro 3.1, you should disable this option. For CEE 3.1, this option should be enabled. Default is enable.

- **CONFIG_RFS_VERSION**

<u>Description</u>

This option is used for RFS version string

<u>Behavioral description</u>

None

<u>Additional notice</u>

- **CONFIG_RFS_PRE_ALLOC**

<u>Description</u>

This option is used for advanced cluster allocation. Current version support maximum 50 numbers for pre clusters allocation.

<u>Behavioral description</u>

Amount of memory used in cluster allocation will be `(CONFIG_RFS_PRE_ALLOC * cluster size)`

<u>Additional notice</u>